

## Netzprogrammierung 6. CORBA I

Prof. Dr.-Ing. Robert Tolksdorf  
Freie Universität Berlin  
Institut für Informatik  
Netzbasierende Informationssysteme  
mailto: [tolk@inf.fu-berlin.de](mailto:tolk@inf.fu-berlin.de)  
<http://www.robert-tolksdorf.de>



[1] © Robert Tolksdorf, Berlin

## Überblick

1. OMG/CORBA
2. IDL
3. Java IDL Mapping

[2] © Robert Tolksdorf, Berlin

## CORBA

- RMI ist sprachgebundene Plattform von Sun
  - **CORBA** (Common Object Request Broker Architecture):
    - offene
    - sprachunabhängige
    - herstellerunabhängige
- Plattform für vernetzte Anwendungen
- Träger: Object Management Group **OMG**
    - „The Object Management Group (OMG) is an open membership, not-for-profit consortium that produces and maintains computer industry specifications for interoperable enterprise applications.“

[3] © Robert Tolksdorf, Berlin

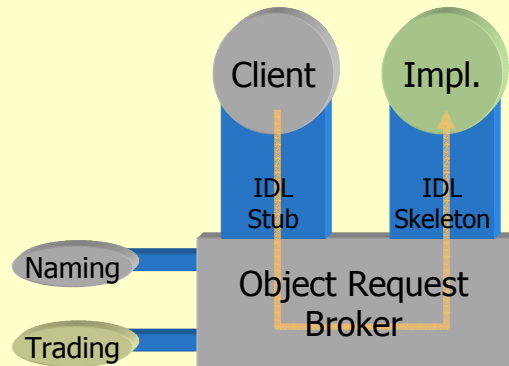
## OMG

- Beispiele von OMG Standards
  - UML – Unified Modelling language
  - CORBA – Common Object Request Broker Architecture
  - CWM – Common Warehouse Metamodel
  - MDA – Model Driven Architecture
- Zusammenarbeit mit ISO
- Mitglieder
  - *alle* relevanten großen und kleinen IT Hersteller
- [www.omg.org](http://www.omg.org)
- OMG-CORBA ist der Industriestandard für die Infrastruktur plattformübergreifender verteilter Anwendungen

[4] © Robert Tolksdorf, Berlin

## OMG

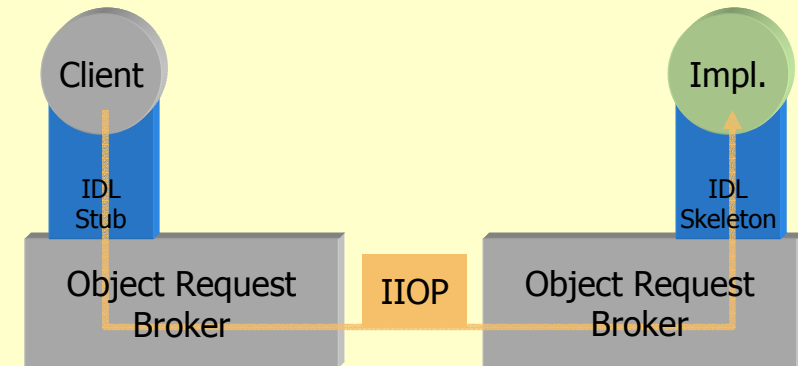
- CORBA Objekte sind Bausteine von Anwendungen
- Sie haben eine typisierte Schnittstelle
- Von der Schnittstelle getrennte Implementierungen in unterschiedlichen Programmiersprachen
- Aufrufe werden durch Object Request Broker transportiert
- Weitere Dienste unterstützen



[5] © Robert Tolksdorf, Berlin

## OMG

- Ortstransparenz wird durch Internet Inter-ORB Protocol IIOP erzeugt



- Interworking zwischen ORBs unterschiedlicher Hersteller möglich

[6] © Robert Tolksdorf, Berlin

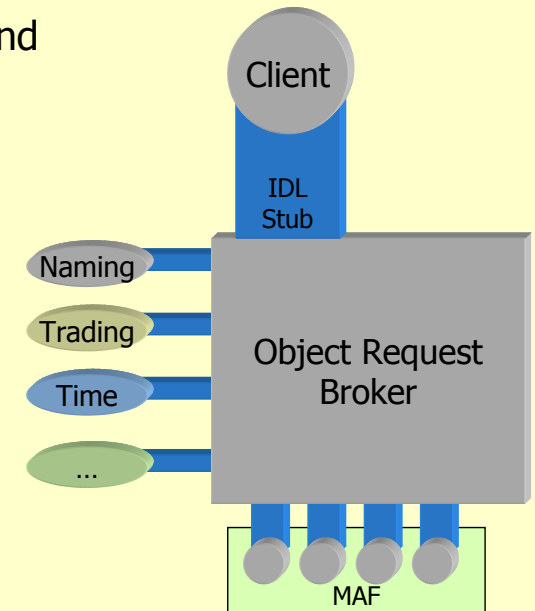
## CORBA Bestandteile

- Interface Definition Language IDL zur Definition von Schnittstellen
- Mappings von IDL zu Programmiersprachen
- APIs zum Objektaufruf (statisch und dynamisch)
- ORB als Transportschicht
- Spezielle Objekte
  - CORBA Services: Verteilt angebotene niedere Dienste
  - Common Facilities: Nützliche Anwendungsbezogene Dienste, die aber nicht als notwendige Dienste angeboten werden müssen. Heutiger Stand: Secure Time, Internationalization, Print Facility, Mobile Agent Facility.

[7] © Robert Tolksdorf, Berlin

## CORBA Produkte

- CORBA Bestandteile sind einzeln erhältlich
  - ORBs
  - IDL Anbindungen
  - Services
  - MAF
- Durch CORBA können diese Komponenten unterschiedlicher Herkunft und Implementierung zusammenarbeiten



[8] © Robert Tolksdorf, Berlin

## Services (Auswahl)

- **Collection Service**  
Provides a uniform way to create and manipulate the most common collections generically.  
bag, collection, heap, iterator, key, ordered, sequence, sequential, set, sorted
- **Query Service**  
Provides query operations on collections of objects.  
collection, iterator, key, ordered, property, query, sequential, set
- **Relationship Service**  
Allows entities and relationships to be explicitly represented. Entities are represented as CORBA objects. The service defines two kinds of objects: relationships and roles.  
cardinality, check out, containment, degree, employment, graph, ownership, reference, relationship, role, type
- **Time Service**  
Enables a user to obtain current time together with an error estimate associated with it.  
event, time, Time Interval Object, timer, TIO, UTC, Universal Time Object, UTO
- Vgl. Java Pakete, z.B. Collections

[9] © Robert Tolksdorf, Berlin

## Services (Auswahl)

- **Naming Service**  
Provides the principal mechanism through which most clients of an ORB-based system locate objects that they intend to use (make requests of).  
binding, client, context, iterator, names, naming, resolve, URL
- Vgl. RMI Registry
- **Trading Object Service**  
The OMG trading object service facilitates the offering and the discovery of instances of services of particular types.  
export, import, lookup, policy, property, query, trader

[10] © Robert Tolksdorf, Berlin

## Services (Auswahl)

- **Persistent State Service**  
Interfaces which present persistent information as storage objects stored in storage homes. Storage homes are themselves stored in datastores, an entity that manages data, for example a database, a set of files, a schema in a relational database.  
catalog, connector, datastore, persistence, PSDL, session, transaction
- **Transaction Service**  
Provides interfaces that combine the transaction paradigm, essential to developing reliable distributed applications, and the object paradigm, key to productivity and quality in application development, together to address the business problems of commercial transaction processing.  
abort, ACID, automicity, commit, committed, completion, durability, isolation, lock, policy, propagation, rollback, security, synchronization, thread, transaction
- **Externalization Service**  
Defines protocols and conventions for externalizing (recording the object's state in a stream of data) and internalizing objects.  
client, file, externalization, node, stream
- Vgl. Java Serialisierung

[11] © Robert Tolksdorf, Berlin

## Services (Auswahl)

- **Concurrency Service**  
Mediates concurrent access to an object such that the consistency of the object is not compromised when accessed by concurrently executing computations.  
concurrent, lock, nested, transaction, unlock
- **Life Cycle Service**  
Life Cycle Service defines services and conventions for creating, deleting, copying and moving objects.  
copy, client, create, delete, factory, move
- **Security Service**  
A security reference model that provides the overall framework for CORBA security.  
access control, accountability, assurance, audit, authentication, certification, DAC, key, MAC, non-repudiation, policy, protection, security, trust
- **Event Service**  
Defines two roles for objects: the supplier role and the consumer role. produce event data and *consumers* process event data. Event data are communicated between suppliers and consumers by issuing standard CORBA requests.  
channel, consumer, event, pull, push, supplier
- **Licensing Service**  
Defines the interfaces that support management of software licenses.  
client, license, policy, producer

[12] © Robert Tolksdorf, Berlin

## Facilities (Auswahl)

### ▪ **Internationalization and Time**

Provide a means for localizing the representation of date, time and number data according to the cultural preferences of users.

additive number, boundary match, collation table, formatter, iterator, locale, numbering system, parameter, pattern, standard match, text

### ▪ **Mobile Agent Facility**

Supports interoperability between various manufacturer's agent systems.

agent, authentication, authority, delegation, naming, profile, registration, security

[13] © Robert Tolksdorf, Berlin

## Überblick / Glossar 1

- Object Management Group *OMG* entwickelt Standards
- Object Management Architecture *OMA* als Standard für verteiltes offenes Objektsystem
- Object Request Broker *ORB* als Laufzeitsystem darin
- Common Object Request Broker *CORBA* als abstrakte Spezifikation von ORBs
- Interface Definition Language *IDL* als Sprache zur Beschreibung von Schnittstellen in OMA
- Internet Inter-ORB Protocol *IIOP* als Protokoll zwischen ORBs

[14] © Robert Tolksdorf, Berlin

## Überblick / Glossar 2

- *ORB* ist die eigentliche Middleware
- *Stub* ist Stellvertreter für entferntes Objekt
- *Skeleton / Adapter* ist Verwaltung des aufrufbaren Objekts
- *Portable Object Adapter POA* als Adapter Muster
- *IDL* beschreibt Schnittstellen
- *IDL Compiler* erzeugt Stub und Skeleton und jeweiliges Interaktion mit ORB in Programmiersprache

[15] © Robert Tolksdorf, Berlin

## Überblick / CORBA Objekte

- Eine mit IDL beschriebene Schnittstelle
- ... kann *verschiedene Implementierungen* ...
- ... in *verschiedenen Programmiersprachen* ...
- ... aus denen sich *verschiedene Exemplare* ergeben...
- ... die als *CORBA Objektreferenz* zugänglich sind ...
- ... und *über ORB aufgerufen* werden können.

[16] © Robert Tolksdorf, Berlin

## Beispiel

## Counter Beispiel

- Zähler Objekt als CORBA Service anbieten und nutzen
- Kann Zähler erhöhen, kann Zählerstand zu einem Parameter addieren und Auskunft über die Anzahl der Änderungen geben
- Notwendig:
  - Schnittstellendefinition
  - Zählerobjekt
  - Registrierung des Servers
  - Nutzung des Servers durch Client
- CORBA ist Teil des Java SDK Standard Edition
- Dort aber nur in Teilen implementiert

## Schnittstellendefinition in IDL

```
module counter {  
  
    typedef struct Info {  
        long value;  
        long counted;  
    } _Info;  
  
    interface Counter {  
        readonly attribute long value;  
        void add(in long value);  
        void addTo(inout long clientValue);  
        Info getInfo();  
    };  
};
```

Eigener Typ

Schnittstelle

Signatur

Namenraum

## Erzeugung von Stubs etc

- >idlj -fall counter.jdl
- Erzeugt in einem Paket counter verschiedene Klassen
- Abbildung Info Typ auf Java-Klasse
  - Info.java: Java Klasse
  - InfoHelper.java: Marshalling der Info Objekte
  - InfoHolder.java: Hilfsklasse für Info als inout Parameter
- Stubs und Skeleton für Counter
  - CounterOperations.java: Counter Interface
  - Counter.java: Typisierung als CORBA Interface
  - CounterHelper.java: Marshalling der Counter Objekte
  - CounterPOA.java: Server Skeleton (Object Adapter)
  - \_CounterStub.java: Stub für Client
- >javac counter/\*.java

## Counter Objekt

```
package counter;
public class CounterServiceImpl extends CounterPOA {
    int value = 0;           // lokale Felder
    int addedCounter = 0;

    public int value() {return value;} // value Attribut herausgeben
    public void add (int value) {      // Zähler erhöhen
        this.value+=value;
        addedCounter++;
    }
    // Zähler zu Parameter addieren
    public void addTo(org.omg.CORBA.IntHolder clientValue) {
        clientValue.value+=value;
    }
    public Info getInfo() {           // Info abliefern
        Info info = new Info();
        info.value=value;
        info.counted=addedCounter;
        return (info);
    }
}
```

[21] © Robert Tolksdorf, Berlin

## Counter Server Programm

```
package counter;
import org.omg.CORBA.*;
import org.omg.CosNaming.*;
import org.omg.PortableServer.*;

public class CounterServer {
    public static void main(String[] args) throws Exception {
        // ORB initialisieren
        ORB orb = ORB.init(args,null);
        // Root Objekt ermitteln und aktivieren
        org.omg.CORBA.Object rootRef =
            orb.resolve_initial_references("RootPOA");
        POA root = POAHelper.narrow(rootRef); // Cast (POA) rootRef
        root.the_POAManager().activate();
    }
}
```

[22] © Robert Tolksdorf, Berlin

## Counter Server Programm

```
// Namensdienst ermitteln
org.omg.CORBA.Object namingRef =
    orb.resolve_initial_references("NameService");
NamingContextExt naming =
    NamingContextExtHelper.narrow(namingRef); // (NamingContextExt)
// Neuen Counter erzeugen
CounterServiceImpl theCounter = new CounterServiceImpl();
// Referenz auf Counter Objekt unter "Counter" registrieren
org.omg.CORBA.Object serviceRef =
    root.servant_to_reference(theCounter);
NameComponent[] path = naming.to_name("Counter");
naming.rebind(path,serviceRef);
// Auf Aufrufe warten
orb.run();
}
```

[23] © Robert Tolksdorf, Berlin

## Counter Client Programm

```
package counter;
import org.omg.CORBA.*;
import org.omg.CosNaming.*;
import org.omg.CosNaming.NamingContextPackage.*;

public class CounterClient {
    public static void main(String[] args) throws Exception {
        // ORB initialisieren
        ORB orb = ORB.init(args,null);
        // Namensdienst auffinden
        org.omg.CORBA.Object namingRef =
            orb.resolve_initial_references("NameService");
        NamingContextExt naming =
            NamingContextExtHelper.narrow(namingRef); // (NamingContextExt)
        // Referenz auffinden
        org.omg.CORBA.Object counterRef =
            naming.resolve_str("Counter");
        Counter counter = CounterHelper.narrow(counterRef);
    }
}
```

[24] © Robert Tolksdorf, Berlin

## Counter Client Programm

```
// Etwas spielen
System.out.println("Wert: "+counter.value());
counter.add(10);
System.out.println("Wert: "+counter.value());
counter.add(10);
System.out.println("Wert: "+counter.value());
IntHolder myInt=new IntHolder(100);
counter.addTo(myInt);
System.out.println("myInt: "+myInt.value);
Info info = counter.getInfo();
System.out.println("Info: "+info.value+" / "+info.counted);
}
}
```

[25] © Robert Tolksdorf, Berlin

## Verteilt ausführen

```
>orbd -ORBInitialPort 1050 -ORBInitialHost localhost
```

```
>java counter.CounterServer -ORBInitialPort 1050 -ORBInitialHost localhost
```

```
athos>java counter.CounterClient -ORBInitialPort 1050
      -ORBInitialHost 160.45.114.204
```

```
Wert: 40
Wert: 50
Wert: 60
myInt: 160
Info: 60 / 6
```

[26] © Robert Tolksdorf, Berlin

## RMI vs CORBA

<i>RMI</i>	<i>CORBA</i>
Eine Sprache	Sprachunabhängig
keine Mappings nötig	Mappings notwendig
Objekte können als Argumente verwendet werden (Weil Klassen nachladbar sind)	Nur Referenzen auf Objekte möglich
Ein Laufzeitsystem	Unterschiedliche Laufzeitsysteme (ORBs) integriert
Proprietär (?) „Eigentümer“: Sun Prozess: JCP	Offener Standard (?) „Eigentümer“: OMG Prozess: OMG

- Zur Klarheit: In Java sind mehrere „Middlewares“ integriert (RMI, CORBA, EJB, ...)

[27] © Robert Tolksdorf, Berlin

## Interface Definition Language IDL

[28] © Robert Tolksdorf, Berlin

## IDL

- IDL ist eine Sprache zur Definition von Schnittstellen
- Sprachunabhängig
- Semantik der Typen definiert
- Zur Nutzung Mapping zu konkreter Sprache notwendig
- OMG definiert solche Mappings für unterschiedlichste Sprachen
- Abbildungsprobleme müssen zur Laufzeit durch Ausnahmen signalisiert werden

[29] © Robert Tolksdorf, Berlin

## IDL Typen (Ausschnitt)

- Einfache Typen
  - Nicht interpretiertes Byte mit 8 Bit Länge  
octet
  - Ganzzahlen mit 16, 32, 64 Bit Länge, mit/ohne Vorzeichen:  
short            unsigned short  
long             unsigned long  
long long        unsigned long long
  - Fließkommazahlen mit einfacher, doppelter und erweiterter Genauigkeit nach IEEE, Festkommazahlen  
float            double            long double            fixed
  - Zeichen für beliebige Zeichen, 8 Bit oder länger  
char             wchar             string                    wstring
  - Wahrheitswert (TRUE, FALSE)  
boolean
  - Typplatzhalter  
any

[30] © Robert Tolksdorf, Berlin

## Zusammengesetzte IDL Typen (Ausschnitt): struct

- Beispiel:

```
struct cumulativeWeatherInfo {
    long start; // Beginning of interval in Unix time() format
    long end;   // End of interval in Unix time() format
    float sunshine; // Hours of sunshine during interval.
    float rainfall; // mm of rainfall during interval.
};
```
- Syntax:

```
(69) <struct_type> ::= "struct" <identifier> "{"
    <member_list> "}"
(70) <member_list> ::= <member>+
(71) <member> ::= <type_spec> <declarators> ";"
```

[31] © Robert Tolksdorf, Berlin

## Zusammengesetzte IDL Typen (Ausschnitt): unions

- Beispiel

```
union U2 switch (char) {
    case 'a':
        long a;
    case 'b':
    case 'c':
        short b;
    default:
        octet c; };
```
- Syntax:

```
(72) <union_type> ::= "union" <identifier> "switch" "(" <switch_type_spec> ")"
    "{" <switch_body> "}"
(73) <switch_type_spec> ::= <integer_type> | <char_type> | <boolean_type> |
    <enum_type> | <scoped_name>
(74) <switch_body> ::= <case>+
(75) <case> ::= <case_label>+ <element_spec> ","
(76) <case_label> ::= "case" <const_exp> ":" | "default" ":"
(77) <element_spec> ::= <type_spec> <declarator>
```

[32] © Robert Tolksdorf, Berlin

## Zusammengesetzte IDL Typen (Ausschnitt): enum

- Beispiel

```
enum color {red, green, blue};
```
- Syntax

```
(78) <enum_type> ::= "enum" <identifier>  
      "{" <enumerator> { "," <enumerator> }* "}"  
(79) <enumerator> ::= <identifier>
```

## Zusammengesetzte IDL Typen (Ausschnitt): array

- Beispiel

```
string StringArray[10];
```
- Syntax

```
(83) <array_declarator> ::=  
      <identifier> <fixed_array_size> +  
(84) <fixed_array_size> ::= "[" <positive_int_const> "]"
```

## Zusammengesetzte IDL Typen (Ausschnitt): sequence

- Beispiel

```
sequence <float>          severalFloats;  
sequence <boolean,10> tenBooleans;
```
- Syntax

```
(80) <sequence_type> ::=  
      "sequence" "<" <simple_type_spec> "," <positive_int_const> ">"  
      | "sequence" "<" <simple_type_spec> ">"
```

## Module und Schnittstellen

- Modul
  - Namensraum für zusammengehörige Schnittstellen
  - Qualifizierter Name: *Modulname::Name*
  - Vergleichbar Paket in Java
- Schnittstelle
  - Getypter Interaktionspunkt für Objekte
  - Vergleichbar einem Interface in Java
- Schnittstelle enthält
  - Konstante
  - Attribute
  - Typdefinitionen
  - Operationen
  - Ausnahmedefinitionen
  - ...

## Module und Schnittstellen

- Beispiel für Objektschnittstelle mit nur einem Attribut

```
module weather {  
    interface weatherdata {  
        struct cumulativeWeatherInfo {  
            long start;  
            long end;  
            float sunshine;  
            float rainfall;  
        };  
    };  
};
```

- Bezeichner: Zeichen, Ziffern, \_
- *Groß-/Kleinschreibung wird nicht unterschieden*

[37] © Robert Tolksdorf, Berlin

## Module und Schnittstellen

- Komplettes Modul mit Typdefinition und Schnittstelle mit Operationen und Attributen

```
module counter {  
  
    typedef struct Info {  
        long value;  
        long counted;  
    } _Info;  
  
    interface Counter {  
        readonly attribute long value;  
        void add(in long value);  
        void addTo(inout long clientValue);  
        Info getInfo();  
    };  
};
```

[38] © Robert Tolksdorf, Berlin

## Syntax

```
<interface_dcl> ::= <interface_header> "{" <interface_body> "}"  
<interface_header> ::=  
    [ "abstract" | "local" ] "interface" <identifier>  
    [ <interface_inheritance_spec> ]  
<interface_inheritance_spec>  
    ::= ":" <interface_name> { "," <interface_name> }*  
<interface_name> ::= <scoped_name>  
<interface_body> ::= <export>*  
<export> ::= <type_dcl> ";;"  
    | <const_dcl> ";;"  
    | <except_dcl> ";;"  
    | <attr_dcl> ";;"  
    | <op_dcl> ";;"
```

[39] © Robert Tolksdorf, Berlin

## Konstante

- Beispiel  
    const maxValue = 2020;
- Syntax  
(27) <const\_dcl> ::=  
    "const" <const\_type> <identifier> "=" <const\_exp>  
(28) <const\_type> ::=  
        <integer\_type>  
        | <char\_type>  
        | <wide\_char\_type>  
        | <boolean\_type>  
        | <floating\_pt\_type>  
        | <string\_type>  
        | <wide\_string\_type>  
        | <fixed\_pt\_const\_type>  
        | <scoped\_name>  
        | <octet\_type>

[40] © Robert Tolksdorf, Berlin

## Attribute

- Beispiel  
attribute float radius;  
readonly attribute position\_t position;
- Syntax  
(85) <attr\_dcl> ::= <readonly\_attr\_spec> | <attr\_spec>  
(104) <readonly\_attr\_spec> ::=  
    "readonly" "attribute" <param\_type\_spec> <readonly\_attr\_declarator>  
(105) <readonly\_attr\_declarator > ::=  
    <simple\_declarator> <raises\_expr>  
    | <simple\_declarator> { "," <simple\_declarator> }\*  
(106) <attr\_spec> ::=  
    "attribute" <param\_type\_spec> <attr\_declarator>  
(107) <attr\_declarator> ::=  
    <simple\_declarator> <attr\_raises\_expr>  
    | <simple\_declarator> { "," <simple\_declarator> }\*

## Typdefinitionen

- Beispiele  
typedef struct Info {  
    long value;  
    long counted;  
} \_Info;  
  
typedef sequence<Book> BookList;

## Operationen

- Beispiel  
• void add(in long value);
- Syntax  
(87) <op\_dcl> ::= [ <op\_attribute> ]  
    <op\_type\_spec> <identifier> <parameter\_dcls>  
    [ <raises\_expr> ] [ <context\_expr> ]  
(88) <op\_attribute> ::= "oneway"  
(89) <op\_type\_spec> ::= <param\_type\_spec> | "void"

## Parameter- und Operationsattribute

- Parameterattribute
    - in  
Parameter wird vom Client zum Server geschickt (by value)
    - out  
Parameter wird vom Server zum Client geschickt (Ergebnis)
    - inout  
Parameter wird in beide Richtungen geschickt (änderbar)
- void add(in long value);  
void addTo(inout long clientValue);
- Operationsattribut
    - oneway
      - Andere Semantic: at-most-once
      - Ergebnis void, keine out Parameter, keine Ausnahme
    - Normalfall ohne Ausnahme: Exactly once
    - Bei Ausnahme: at-most-once

## Ausnahmen

- In CORBA sind zwei Arten Ausnahmen definiert
  - Standardausnahmen:

```
#define ex_body {unsigned long minor; completion_status completed;}
module CORBA {
  exception UNKNOWN ex_body; // the unknown exception
  exception BAD_PARAM ex_body; // an invalid parameter was passed
  exception NO_MEMORY ex_body; // dynamic memory allocation failure
  exception IMP_LIMIT ex_body; // violated implementation limit
  exception COMM_FAILURE ex_body; // communication failure
  ...
}
```
  - Deklarierte anwendungsspezifische Ausnahmen

[45] © Robert Tolksdorf, Berlin

## Ausnahmen

- Ausnahmedeklaration
  - Beispiel
    - exception overspending {long amount;};
  - Syntax
    - (86) <except\_dcl> ::=  
"exception" <identifier> "{" <member>\* "}"
- Raises Klausel bei Methodendeklaration:

```
interface Wallet {
  exception overspending {long amount;};
  void pay(long prize) raises(overspending);
};
```

[46] © Robert Tolksdorf, Berlin

## Java IDL Mapping

[47] © Robert Tolksdorf, Berlin

## IDL Mapping für Java

- Einfach Typen zu geeigneten Primitiven:

boolean	boolean
char, wchar	char
string, wstring	String
octet	byte
short, unsigned short	short
long, unsigned long	int
long long, unsigned long long	long
float	float
double	double
fixed	java.math.BigDecimal

[48] © Robert Tolksdorf, Berlin

## IDL Mapping für Java

- IDL

```
void basicints(in short s, in unsigned short us,
               in long l, in unsigned long ul,
               in long long ll, in unsigned long long ull);
void basicfloats(in float f, in double d);
void basicchars(in char c, in wchar wc,
               in string s, in wstring ws);
void basicmisc (in octet o, in boolean b);
```
- Java

```
void basicints (short s, short us, int l, int ul, long ll, long ull);
void basicfloats (float f, double d);
void basicchars (char c, char wc, String s, String ws);
void basicmisc (byte o, boolean b);
```

[49] © Robert Tolksdorf, Berlin

## sequence -> Array

- Listen zu Felder
- IDL

```
typedef sequence <float>          severalFloats;
typedef sequence <boolean,10> tenBooleans;
void mapsequence(in severalFloats sf, in tenBooleans tb);
```
- Java

```
void mapsequence (float[] sf, boolean[] tb);
```

[50] © Robert Tolksdorf, Berlin

## struct -> Klasse

- Verbünde zu Objekte mit Attributen
- IDL

```
typedef struct Info {
    long value;
    long counted;
} Info;
```
- Java

```
public final class Info implements org.omg.CORBA.portable.IDLEntity {
    public int value = (int)0;
    public int counted = (int)0;

    public Info () { } // ctor

    public Info (int _value, int _counted)
    {
        value = _value;
        counted = _counted;
    } // ctor
} // class Info
```

[51] © Robert Tolksdorf, Berlin

## enums -> Klasse

- Aufzählungen zu Aufzählungsobjekte
- IDL

```
typedef enum color {red, green, blue} _color;
void mapenum(in color c);
```

[52] © Robert Tolksdorf, Berlin

## enums -> Klasse

- Java

```
void mapenum (types.typedestsPackage.color c);

private int __value;
private static int __size = 3;
private static types.typedestsPackage.color[] __array =
    new types.typedestsPackage.color [__size];
public static final int _red = 0;
public static final types.typedestsPackage.color red=new
    types.typedestsPackage.color(_red);
public static final int _green = 1;
...
public int value () { return __value; }
public static types.typedestsPackage.color from_int (int value)
{if (value >= 0 && value < __size) return __array[value];
 else
     throw new org.omg.CORBA.BAD_PARAM ();
}
protected color (int value)
{__value = value;
 __array[__value] = this;
}
```

[53] © Robert Tolksdorf, Berlin

## union -> Klasse

- Unions nach Unionobjekte
- IDL

```
typedef union borc switch (short) {
case 0: short b;
case 1: long c;
} _borc;
void mapunion(in borc b);
```

[54] © Robert Tolksdorf, Berlin

## union -> Klasse

- Java

```
void mapunion (types.typedestsPackage.borc b);

public final class borc implements org.omg.CORBA.portable.IDLEntity
{ private short __b;
  private int __c;
  private short __discriminator;
  private boolean __uninitialized = true;

  public borc () { }
  public short discriminator () {... }
  public short b () {...}
  public void b (short value) {... }
  public int c () {... }
  public void c (int value). {... }
  ...
```

[55] © Robert Tolksdorf, Berlin

## Operationen

- Operationen nach Methoden
- Exceptions nach Exceptions
- Parameterübergabe
  - in Parameter: Normale Werteparameter in Java
  - out und inout Parameter: keine Entsprechung in Java
  - Werden durch Holder-Klassen realisiert
  - Parameter sind dann Objekte dieser Klassen

[56] © Robert Tolksdorf, Berlin

## Holder Klassen

- Vordefiniert für vordefinierte Typen
- org.omg.CORBA.IntHolder:
  - Field Summary
    - float value  
The float value held by this FloatHolder object.
  - Constructor Summary
    - FloatHolder()  
Constructs a new FloatHolder object with its value field initialized to 0.0.
    - FloatHolder(float initial)  
Constructs a new FloatHolder object for the given float.
- Entsprechend int, long etc.etc.

[57] © Robert Tolksdorf, Berlin

## Aus Counter Beispiel

- Aus IDL:  
void addTo(inout long clientValue);
- Beim Serverobjekt:  
// Zähler zu Parameter addieren  
public void addTo(org.omg.CORBA.IntHolder clientValue) {  
    clientValue.value += value;  
}
- Beim Klientenobjekt Parameter erzeugen:  
IntHolder myInt = new IntHolder(100);  
counter.addTo(myInt);  
// veränderter Wert in myInt.value

[58] © Robert Tolksdorf, Berlin

## Holder Klassen

- Automatisch erzeugt bei eigenen Typen

```
package counter;
/**
 * counter/InfoHolder.java .
 * Generated by the IDL-to-Java compiler (portable), version "3.1"
 * from counter.idl
 * Mittwoch, 22. Oktober 2003 15.15 Uhr CEST
 */
public final class InfoHolder implements org.omg.CORBA.portable.Streamable
{
    public counter.Info value = null;

    public InfoHolder () { }

    public InfoHolder (counter.Info initialValue)
    { value = initialValue; }

    [...]
}
```

[59] © Robert Tolksdorf, Berlin

## Literatur

- ISO/IEC 14750:1999 Information technology - Open Distributed Processing - Interface Definition Language. JTC 1/SC 7. 1999.
- OMG. CORBA 3.0 Spezifikation.  
[http://www.omg.org/technology/documents/formal/corba\\_2.htm](http://www.omg.org/technology/documents/formal/corba_2.htm)  
1150 Seiten!
- Open Source CORBA Implementierungen:  
[http://sourceforge.net/softwaremap/trove\\_list.php?form\\_cat=51](http://sourceforge.net/softwaremap/trove_list.php?form_cat=51)
- Dokumentation des JSDK 1.4SE

[60] © Robert Tolksdorf, Berlin

## Zusammenfassung

## Zusammenfassung

1. **OMG/CORBA**
  1. Sprachunabhängiges Objektmodell
  2. ORB als Transportschicht
  3. IDL für Schnittstellen
  4. Services, Facilities als definierte Dienstobjekte
2. **IDL**
  1. Einfache Typen
  2. Zusammengesetzte Typen
  3. Module und Schnittstellen
  4. Operationen
3. **Java IDL Mapping**
  1. IDL Konzepte teilweise direkt abgebildet
  2. IDL Konzepte teilweise auf Helferklassen abgebildet